EE392C: Final Discussion & Brainstorming

Today’s Menu

• Review our in-class discussions
  ▪ What are the major lessons in each topic/overall?
  ▪ Any important topics we missed?

• Brainstorming
  ▪ What are the next steps in each topic/overall?

• Conclusions & predictions
  ▪ Future of parallel architecture (CMPs)?
  ▪ Future of polymorphic CMPS?
  ▪ Future division of labor between HW & SW?

Reminder of Topics

• Architecture
  ▪ Streams/vectors, (speculative) multithreaded, polymorphic

• Applications
  ▪ Media, enterprise, networking, security, AI, verification,…

• Parallel programming models & compilers

• Virtual machines & on-line profiling

• Dynamic compilation

• Fault-tolerance & reliability

• Machine learning techniques & system design

Applications: Lessons

• Emerging applications have a diverse set of characteristics
  ▪ No single architecture best matches all applications
  ▪ Opportunity for polymorphic HW & SW

• In some cases, you need to higher level understanding of the algorithms to optimize and parallelize an application
  ▪ Optimizations may be possible after replacing an algorithm
  ▪ But this can make code architecture specific
  ▪ Use of libraries or templates can reduce this risk
**Architecture: Lessons**

- Parallel architectures are successful only when you have a good software story
- Simple architectures are often sufficient for several application domains
- The granularity argument: fine-grain Vs. coarse-grain
  - Finer grain gives you better flexibility and is easier to build
  - But it typically requires more work in software (programming model or compiler)
  - There is not free lunch...
- Not trivial to tune architecture parameters for CMPs
  - Automated methods (e.g. genetic programming) can be useful
- Important issues:
  - Power, so keep things simple
  - HW for profiling, understand where your cycles are going

**Programming Models & Compilers: Lessons**

- Keep programming model close to current languages
  - Start with C and move forward from that
    - Add features that make it easier to analyze for the compiler
    - But leverage existing code base
  - Keep sequential semantics to make easier to port to sequential machines
- New programming models require
  - Large benefits to motivate change
  - Solid development tools
- Unresolved issue
  - Use a single parallel model & provide libs/pragmas for application specific optimizations
  - Use different languages optimized for each application domain
    - It’s not a huge number but it is not one…

**Architecture: Next Steps**

- Managing HW configurations in polymorphic CMP
  - Who is responsible for selecting the configuration?
    - Fixed by the programming model (today)
    - Selected by the static compiler (but >1 time per program)
    - Change dynamically (dynamic compiler, VM, …)
  - Which from the many configuration options are important/useful and must be support?
- What is the theoretical limit for polymorphic CMPs?
  - Efficiency = f(performance, power, cost, …)
- Scalability
  - How does the architecture & programming model scale over the next decade?
- Architectural support for grouping multiple small cores into a larger core
  - Synchronization, code generation, memory system
  - Focusing on DLP & TLP

**Programming Models & Compilers: Lessons**

- Ideally, hide from programmer
  - All architecture specific features
    - Parallel resources, memory hierarchy
  - Synchronization, memory management
    - Programmer can be wrong sometimes
- What should the programmer tell us?
  - Computation-data affinity, (un)alias info, task parallelism
  - Hints for optimizations + mechanisms for recovery
  - Maybe we can replace most of user info with profiling info
Programming Models & Compilers: Next Steps

- Separate parallel programming model from parallel architecture model
  - Bridge the gap between these two in a VM
- Parallel languages that are easy to analyze for races, deadlocks, etc
  - Template/library-based compilation/parallelization
  - Transactional programming model

Virtual Machines & Profiling: Lessons

- VMs good for fault-tolerance (fast recovery)
- Good for compatibility & porting
  - Port old apps to new archs & the reverse
- VMs may be merged within OS
  - Current VM services (dynamic translation etc) will go into OS
- Maybe OSs will become thinner and look like VMs
- Profiling is easy but summarizing data is difficult

Virtual Machines & Profiling: Next Steps

- Analyze apps to find out what info must be collected during profiling
  - Important parameters, etc
  - How should we summarize this info?
- Polymorphic VMs
- What is the CMP interface exposed by a VM
  - SMP, MP, polymorphic???
  - VM can present polymorphic API on top of a regular architecture

Dynamic Compilation: Lessons

- Dynamic optimizations important as we mostly use OO languages & several pieces of the application are in dynamically-linked libraries
- Dynamic optimizations are important for applications that can be customized for specific data-patterns
Dynamic Compilation: Next Steps

- Statically-guided dynamic compilation
- Increase the scope of dynamic optimizations (e.g., parallelization)
  - Watch out for overheads
- Select configuration of polymorphic CMP with dynamic compiler

Fault Tolerance: Lessons

- Use as much redundancy as possible
  - Redundancy in computation (& voting)
  - Use spares
- Isolate everything
- Can make it fault-tolerance configurable
  - To reduce some of the cost
  - Users that don’t care about fault tolerance can disable it and use redundant cores for regular computation

Machine Learning & Systems: Lessons

- Not obvious what/how much to learn
  - To little info, may not be enough to optimize
  - To much info, may be too difficult to analyze/use/learn without significant overheads

Machine Learning & Systems: Next Steps

- Machine learning & scheduling problems
  - Within the run-time you can use fairly complicated ML algorithms
Overall Conclusions

• Always do as much analysis and optimization as you can in SW before passing binaries to HW
  ▪ Hints for dynamic compilation
  ▪ Optimize for speculative multithreading

Overall Predictions

• CMPs are here to stay
  ▪ But the jury is still out on reconfigurability
  ▪ It will take more than 5 years to get the SW story right for CMPs
• Performance will not be only major concern
  ▪ Power, reliability, cost, software development effort
  ▪ Motivation for simplicity and for revisiting multiple layers of abstraction
• Hardware-software codesign will be common
  ▪ Complement capabilities of each component
  ▪ Hide weaknesses of each component
• Better dynamic compilation techniques will be common

CPUs Vs. ASICs/ASIPs

• Even with general purpose processors we will have specialized coprocessors for some common tasks
  ▪ E.g. crypto
• Time to market constraints may push us towards domain-specific CPUs instead of ASICs
• For apps that do not stress the limits of technology, we will use CPUs more & more
  ▪ Cost and time to market advantages
• From the technical point of view, communication cost can be the determining factor for using a ASIP Vs a CPU for some application